MULTIMEDIA UNIVERSITY

STUDENT ID NO

SEAT NO: ____________

SECTION: TC01 / TC02

MULTIMEDIA UNIVERSITY

FINAL EXAMINATION

TRIMESTER 1, 2017/2018

TCP1201 – OBJECT-ORIENTED PROGRAMMING AND DATA STRUCTURES
(All sections / Groups)

25 OCTOBER 2017
2:30 p.m. – 4:30 p.m.
(2 Hours)

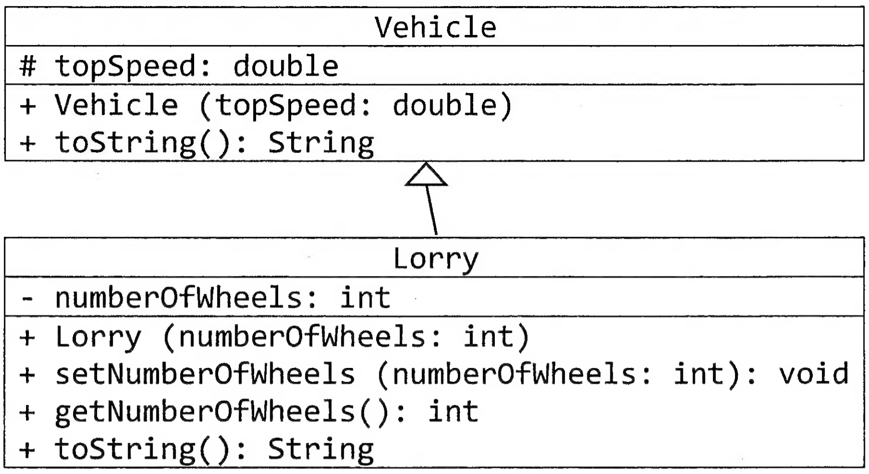
Question	Mark
1	
2	
3	
4	
Total	

INSTRUCTIONS TO STUDENTS

1. This Question paper consists of 15 pages with 4 Questions only.
2. Attempt all **FOUR** questions. All questions carry equal marks and the distribution of the marks for each question is given.
3. Please write all your answers in **this Question Paper**.

Question 1

The following UML Class Diagram is provided.


Vehicle
# topSpeed: double
+ Vehicle (topSpeed: double)
+ toString(): String
Lorry
- numberOfWheels: int
+ Lorry (numberOfWheels: int)
+ setNumberOfWheels (numberOfWheels: int): void
+ getNumberOfWheels(): int
+ toString(): String

a. Explain briefly the relationship between Vehicle class and Lorry class. [3 marks]

b. Does **function overriding** occur in the two classes? If yes, state the **name** of the method that has been overridden. [2 marks]

Continued...

c. **Implement** both the Vehicle class and Lorry class based on the UML Class Diagram above and the main method below. [16 marks]

```
public static void main (String[] args) {
  Vehicle v = new Vehicle (50);
  System.out.println (v);
  Vehicle l = new Lorry (100, 12);
  System.out.println (l);
}
```

Sample run:

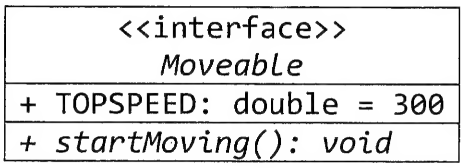
```
Vehicle: topSpeed = 50.0
Lorry: topSpeed = 100.0, number of wheels = 12
```

Continued...

Continued...

d. Provide a reason for a class to be declared **abstract**. [1 mark]

e. The following UML Class Diagram is provided. Write a **declaration** for the Moveable interface. [3 marks]



Continued...

Question 2

a. State the **output** of the program below. [2 marks]

```
class TestStatic {
  private int a = 10;
  private static int b = 10;
  public void doubleUp() {
    a *= 2;
    b *= 2;
  }
  public void print() {
    System.out.println ("a = " + a + ", b = " + b);
  }
  public static void main (String[] args) {
    TestStatic t1 = new TestStatic();
    TestStatic t2 = new TestStatic();
    t1.doubleUp();
    t1.doubleUp();
    t2.doubleUp();
    t1.print();
    t2.print();
  }
}
```

b. When should **aggregation** be used instead of **inheritance**, and **vice versa**? Give an **example** for each case. [4 marks]

Continued...

c. The following incomplete program is provided. The program fails to work because the Product class does not implement a particular method. State the **name** of the missing method and provide an **implementation** for the method. [8 marks]

```
class Product implements Comparable<Product> {
  private String name;
  private double price;
  public Product (String name, double price) {
    this.name = name;
    this.price = price;
  }
  public static void main (String[] args) {
    Product[] products = { new Product("TV", 1000),
                           new Product("Cake", 10),
                           new Product("Book", 80) };
    java.util.Arrays.sort (products);
  }
}
```

Continued...

d. The following code snippet is provided. It determines whether a person is an adult or minor based on the age given. A person is an adult if he/she is 18 years old or more, and is a minor if less than 18 years old. However the program has an error. It considers a negative age as a minor. Use **exception handling** to handle the error. Throw an **IllegalArgumentException** and output "Age cannot be negative" if a negative age is entered. [5 marks]

```
Scanner input = new Scanner(System.in);
System.out.print ("Enter your age: ");
int age = input.nextInt();
if (age < 18)
  System.out.println ("Minor");
else
  System.out.println ("Adult");
```

Sample run 1:
```
Enter your age: 18
Adult
```

Sample run 2:
```
Enter your age: 11
Minor
```

Sample run 3:
```
Enter your age: -1
Age cannot be negative.
```

Continued...

e. Write a **recursive** method that performs the following sum series. [6 marks]

$$sum(i) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{i}$$

Continued...

Question 3

a. Convert the following raw (non-generic) version of the display method to a **generic** version. The display method displays all elements in the list. [2 marks]

```
public void display (Object[] list) {
  for (int i = 0; i < list.length; i++)
    System.out.print(list[i] + " ");
}
```

b. State **two main differences** between a **stack** and a **queue**. [4 marks]

c. Is it more efficient to implement a queue using a linked list or an array list? Explain your answer. [4 marks]

Continued...

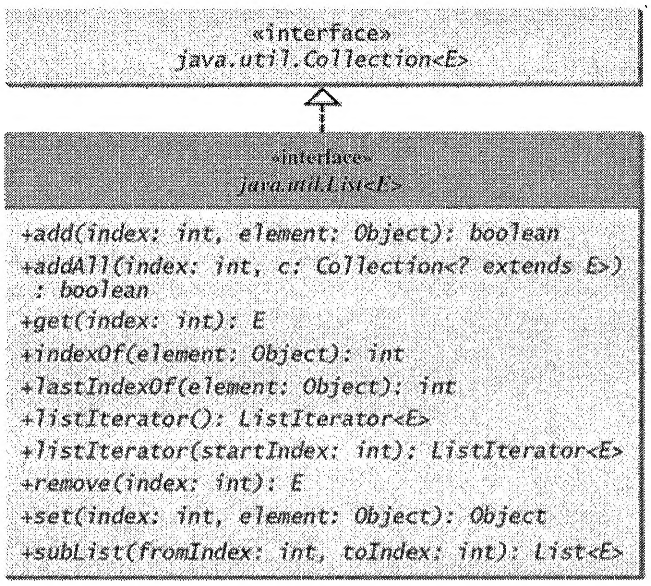
d. The following shows the definition of the **Stack** class. Provide the implementation for the **push** method; the method that inserts an element at the top of the stack, and the **pop** method; the method that removes an element from the top of the stack

[6 marks]

```
import java.util.*;

public class GenericStack<E> {
  private ArrayList<E> list = new ArrayList<>();

  public int getSize() {
    return list.size();
  }

  public E peek() {
    return list.get(getSize() - 1);
  }

  public void push(E o) {






  }

  public E pop() {








  }

  public boolean isEmpty() {
    return list.isEmpty();
  }
}
```

Continued...

e. Using the **LinkedList** class (provided by **java.util** library) with the specification as defined in the UML diagram below, write a program that creates a **LinkedList** instance and store 10 random integers into the linked list in a sorted manner. The program then displays the integers in ascending order. Hint: You may use **Collections.sort** method. [9 marks]


«interface»
java.util.Collection<E>
«interface»
java.util.List<E>
+add(index: int, element: Object): boolean
+addAll(index: int, c: Collection<? extends E>) : boolean
+get(index: int): E
+indexOf(element: Object): int
+lastIndexOf(element: Object): int
+listIterator(): ListIterator<E>
+listIterator(startIndex: int): ListIterator<E>
+remove(index: int): E
+set(index: int, element: Object): Object
+subList(fromIndex: int, toIndex: int): List<E>

```
public class SortedIntegerList {
  public static void main(String[] args) {
```

```
  }
```

Continued...

Question 4

a. The following program consist of a generic class, **AnimalHouse** and three concrete classes, **Animal, Dog** and **Cat**.

```
class AnimalHouse<E> {
    private E animal;
    public void setAnimal(E x) {
        animal = x;
    }
    public E getAnimal() {
        return animal;
    }
}
class Animal{
}
class Cat extends Animal {
}
class Dog extends Animal {
}
```

For the following code snippets, identify whether the code **compiles with errors** (answer = **YES**) or **compile without errors** (answer = **NO**).

[4 marks]

YES / NO

```
(i)   AnimalHouse<Animal> house = new AnimalHouse<Cat>();

(ii)  AnimalHouse<Dog> house = new AnimalHouse<Animal>();

(iii) AnimalHouse<Cat> house = new AnimalHouse<Cat>();

(iv)  AnimalHouse<?> house = new AnimalHouse<Cat>();
      house.setAnimal(new Cat());
```

Continued...

b. (i) Construct a **binary search tree** (show one per insertion) by inserting the following numbers one after another: **9, 10, 8, 2, 5, 12, 1**

[7 marks]

(ii) Provide the pre-order, in-order and post-order traversal for the tree in (i).

[6 marks]

Continued...

d. The following is an incomplete program. It is a program that extracts the **unique words** from the *quote* string and displays them in **ascending** order. Complete the implementation of the program. Below is the sample run of the program.

[8 marks]

Sample run:

```
Sorted tree set: [but, forget, hurts, it, never, taught, what, you]
```

```
import java.util.*;

public class Mystery {
  public static void main(String[] args) {
    String quote = "Forget what hurts you "
                       + " but never forget what it taught you";
    // Create an empty treeset
```

```
    // Split the quote into the "words" array
    String[] words = quote.split("[\\s+\\p{P}]");

    // Loop through the "words" array and store the unique words into
    // the treeset that you created above and print out the tset.
```

```
  }
}
```

End of Paper